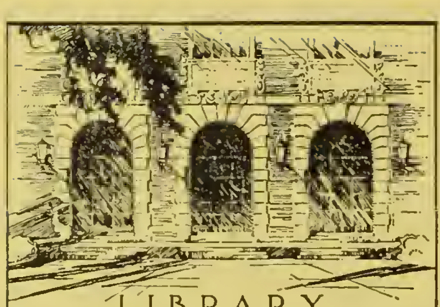

Report No. 275

COO-1018-1152

THE LOGICAL DESIGN OF A CLASS OF LIMITED CARRY-BORROW PROPAGATION ADDERS*

by

Richard T. Borovec

August 1, 1968

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

Report No. 275

THE LOGICAL DESIGN OF A CLASS OF LIMITED CARRY-BORROW PROPAGATION ADDERS*

by

Richard T. Borovec

August 1, 1968

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under contract NSF GP-4636 and Contract AT(11-1)-1018 with the U.S. Atomic Energy Commission and the Advanced Research Projects Agency, and was submitted in partial fulfillment for the degree of Master of Science in Electrical Engineering, 1968.

ACKNOWLEDGMENT

The author would like to thank his advisor, Professor J. E. Robertson, for his helpful advice and guidance in the preparation of this thesis.

The author would also like to thank Miss Colette Portugal for her capable typing and Mr. John Otten for the drawing of the figures in the thesis.

TABLE OF CONTENTS

1. INTRODUCTION

It has been recognized for some time[†] (3,5) that one of the ways of increasing the speed of arithmetic operations in a digital computer is to introduce redundancy into the arithmetic unit, i.e. using k digital values in radix r where $k > r$. Redundancy allows such techniques as multiplier recoding to increase the shift average and the possibility of performing division using a truncated divisor and partial remainder in a model division scheme.

The arithmetic unit of the Illiac III computer (1,2) uses four redundant subtracters in cascade to perform, effectively, radix 256 arithmetic. This means that fewer operations are necessary since larger parts of the operands may be operated on at once. The subtracter used in this arithmetic unit, the block diagram of which is shown in Figure 1, allows one redundant input and one conventional input. In Figure 1 an asterisk represents a redundant digit while a variable standing alone represents a conventional digit.

The basis for this paper is the work by Rohatsch (8) on the algebraic structure of adders and the work by Robertson (6) on the design procedures for redundant adders.

The purpose of this thesis is to determine the logical structure of a class of redundant adders which allow both the addend and the augend to be expressed redundantly. Rohatsch has shown that such adders must consist of three "levels", i.e. three transformations are necessary to map the input digit set onto the output digit set.

[†]Numbers in parentheses refer to List of References at the end of the paper.

Two different design procedures were used and these will be discussed in the next chapter. Chapter three will give the results of the designs of the normalized adder; Chapter four the symmetric adder and subtracter. Chapter five will describe a logical design problem, believed to be new, encountered in the design procedure, the problem of coupled don't cares. A systematic method of assigning covers will also be introduced. Finally, the results will be summarized in Chapter six.

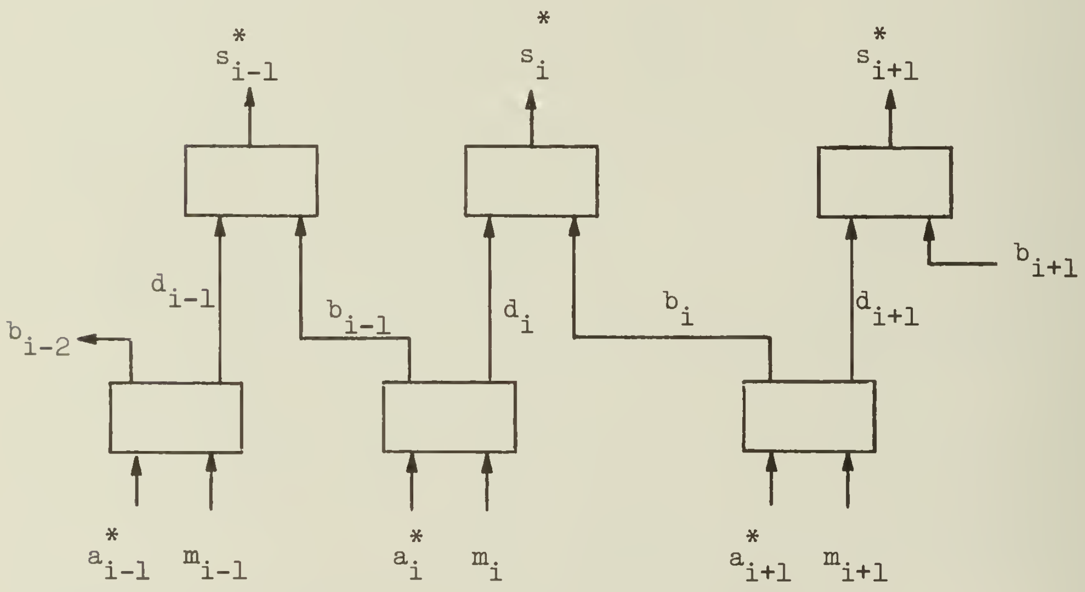

Figure 1. Illiac III Signed Digit Subtracter

2. DESIGN PROCEDURE

The following notation and assumptions will be used for the rest of this thesis: i) x_i^* represents a redundant digit while x_i represents a conventional digit, ii) numbers with overbars, e.g. $\overline{1}$, $\overline{2}$, etc., are negative numbers while unsigned numbers are positive, iii) all designs to be considered will be radix two devices with minimal redundancy, i.e. a redundant digit will be able to assume one of $r + 1$ or three possible values, and iv) the format for expressing the input digits will be the same as the format for expressing the output digits. This is the most interesting case as it allows the output to be used as an input; a property useful in the implementation of recursive processes such as multiplication and division.

Since the designs under investigation are of radix two with minimal redundancy, two variables, i.e. four states, are necessary to represent the possible values a digit may assume. Robertson (6) has shown that there exist only nine distinct ways, under permutation and negation, to assign three values to four states (two binary digits). This implies that there are nine possible designs for each structure type.

Two methods of approaching the logical design were used. The first was to recognize that it was possible to add another level to the design for the two level structures which already existed (7). Using Rohatsch's idea this corresponds to an additional transformation from the new digit set (two redundant inputs) to the original digit set (one redundant input, one conventional input). This entailed the design of a box as shown in Figure 2.

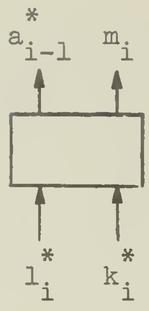

Figure 2

The inputs to the box at the ith digital position were redundant digits l^*_i and k^*_i while the outputs were a^*_{i-1} and m_i. The three redundant digits l^*_i, k^*_i, and a^*_{i-1} were of the same format. The outputs of the box were used as inputs to the existing structures. This design procedure will henceforth be known as procedure A.

The second procedure, suggested by Robertson (7), consisted of designing the lower two levels of the adder without considering the format of interstage signals. This design allows two redundant inputs and gives three binary outputs which are format independent. The structure for this procedure is shown in Figure 3. The second procedure, procedure B, uses the top levels of the adders which have been designed.

The complete structures for procedure A and procedure B are illustrated in Figures 4 and 5 respectively.

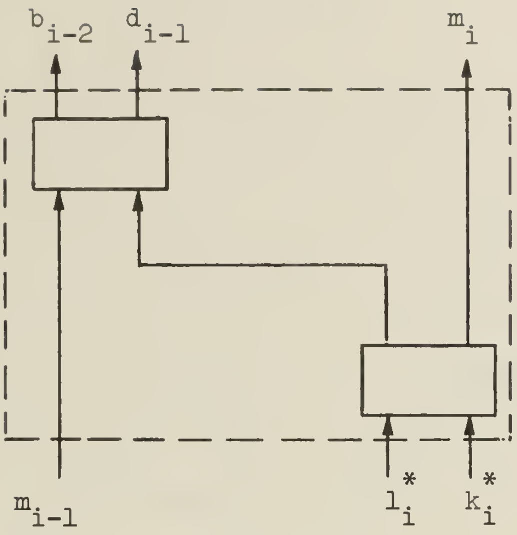

Figure 3

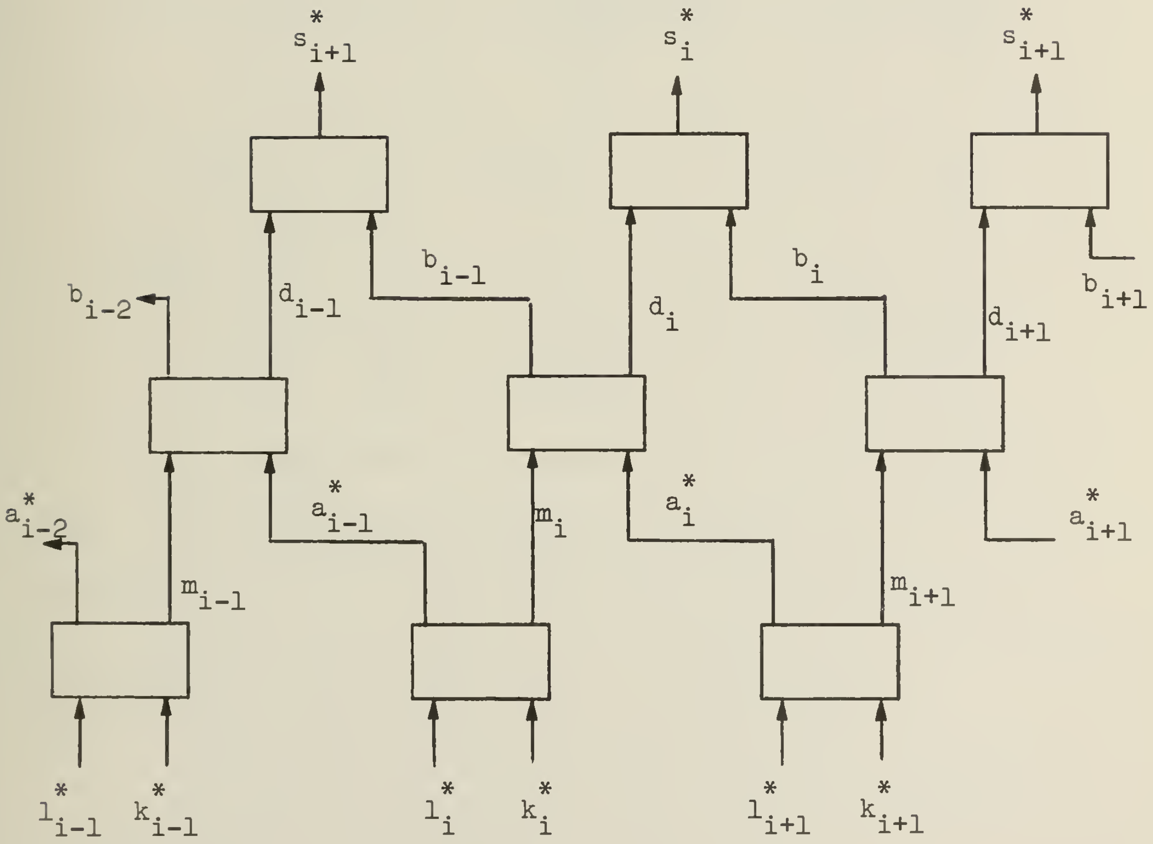

Figure 4. Procedure A Structure

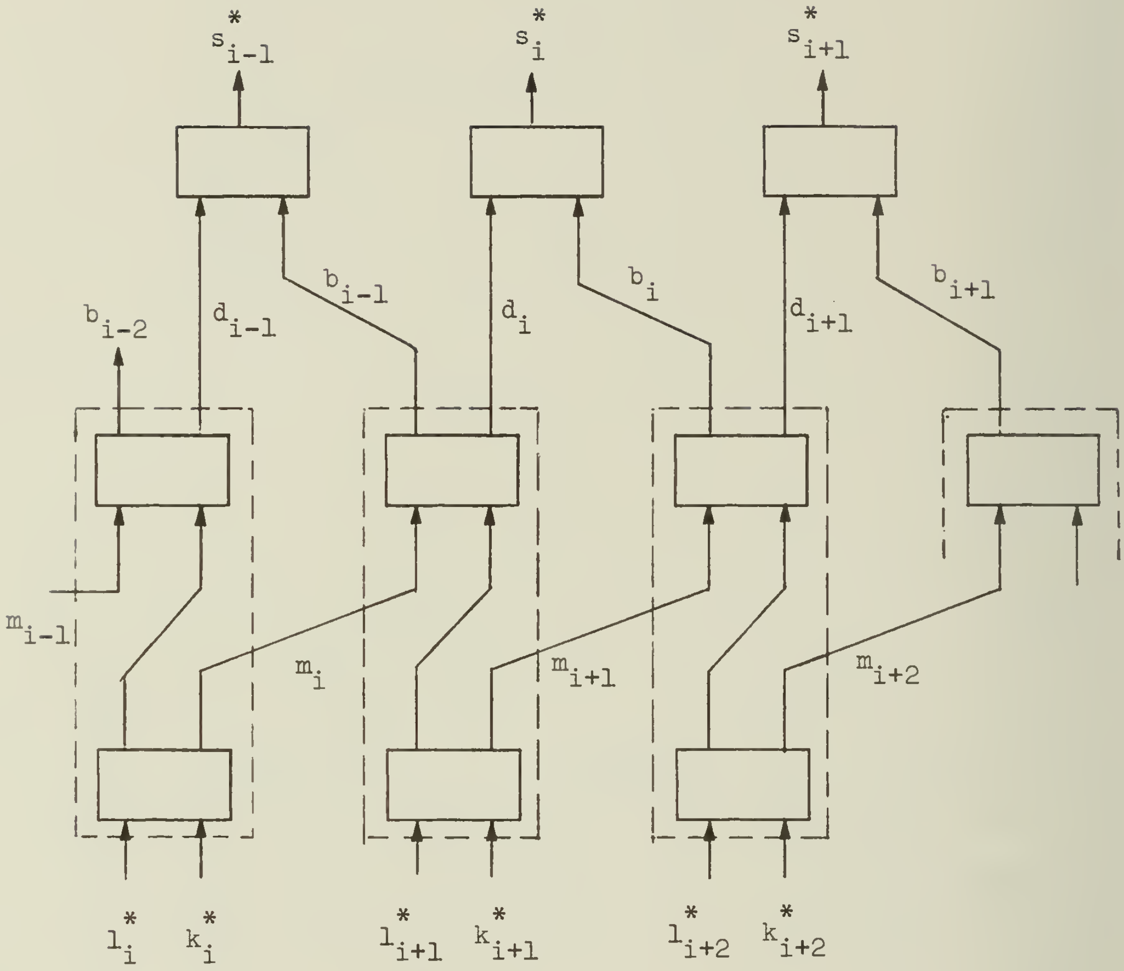

Figure 5. Procedure B Structure

3. NORMALIZED ADDER

The normalized adder is the first structure to be designed using Robertson's deterministic procedure (6) as it uses the normalized digit set {0,1,2}. The adder is defined by the values the variables may assume, namely a_i^*, l_i^*, k_i^*, s_i^* {0,1,2}; b_i, d_i, m_i {0,1}. The format assignments for the adder are given in Table 1. In the designs a Greek letter and a Roman letter are used to denote the first and second variable in the assignment respectively. Thus a_i^* is given by $\alpha_i a_i$.

State	Digital Values								
	0	0	0	0	D.C.	0	0	1	2
	1	1	1	1	1	1	1	1	1
	2	D.C.	0	2	0	1	2	0	0
	0	2	2	D.C.	2	2	2	2	2
Design Number	1	2	3	4	5	6	7	8	9

Table 1. Format Assignments for Normalized Adder

The Boolean equations for design procedure A are given in Table 2. Design procedure B was not used in the case of the normalized adder. The use of the normalized adder to perform subtraction is not a trivial matter and thus it was deemed that the other designs (symmetric adder and subtracter) were of more interest.

In Table 2 designs 2, 3, 4 and 8 appear to be minimal in that they require the fewest gates to implement.

1. $\alpha_{i-1} = \overline{\lambda}_i \oplus l_i$

$a_{i-1} = \kappa_i \overline{k}_i \vee (\lambda_i \oplus l_i)(\kappa_i \vee \overline{k}_i)$

$m_i = \overline{\kappa}_i k_i \oplus \overline{\lambda}_i l_i$

$d_i = m_i \oplus \overline{\alpha}_i a_i$

$b_{i-1} = \alpha_i \overline{a}_i \vee \overline{\alpha}_i a_i m_i$

$\sigma_i = b_i d_i$ or $\sigma_i = \overline{b_i \oplus d_i}$

$s_i = b_i \oplus d_i$ $\quad$ $s_i = b_i \vee d_i$

2. $\alpha_{i-1} = \lambda_i l_i \kappa_i k_i$

$a_{i-1} = l_i k_i \vee \kappa_i k_i \vee \lambda_i l_i$

$m_i = \overline{\lambda}_i l_i \oplus \overline{\kappa}_i k_i$

$d_i = m_i \oplus \overline{\alpha}_i a_i$

$b_{i-1} = \alpha_i \vee a_i m_i$

$\sigma_i = b_i d_i$

$s_i = b_i \vee d_i$

3. $\alpha_{i-1} = \lambda_i l_i \kappa_i k_i$

$a_{i-1} = l_i k_i \vee \kappa_i k_i \vee \lambda_i l_i$

$m_i = \overline{\lambda}_i l_i \oplus \overline{\kappa}_i k_i$

$d_i = m_i \oplus \overline{\alpha}_i a_i$

$b_{i-1} = a_i(\alpha_i \vee m_i)$

$\sigma_i = b_i d_i$

$s_i = b_i \vee d_i$

Table 2. Boolean Equations for Normalized Carry-Save Adders

4. $\alpha_{i-1} = \lambda_i \kappa_i$

$a_{i-1} = \lambda_i \oplus \kappa_i \vee l_i k_i$

$m_i = l_i \oplus k_i$

$d_i = a_i \oplus m_i$

$b_{i-1} = \alpha_i \vee a_i m_i$

$\sigma_i = b_i d_i$

$s_i = b_i \oplus d_i$

5. $\alpha_{i-1} = \lambda_i l_i \kappa_i k_i \vee \overline{l}_i(\overline{\kappa}_i \vee \overline{k}_i) \vee \overline{\lambda}_i \overline{k}_i$

$a_{i-1} = l_i k_i \vee \kappa_i k_i \vee \lambda_i l_i$

$m_i = \lambda_i \oplus \kappa_i$

$d_i = \overline{\alpha_i \oplus m_i}$

$b_{i-1} = a_i(\alpha_i \vee m_i)$

$\sigma_i = \overline{b_i \oplus d_i}$

$s_i = b_i \vee d_i$

6. $\alpha_{i-1} = \lambda_i l_i \kappa_i k_i \vee (\overline{\kappa}_i \vee \overline{k}_i)\ (\lambda_i \oplus l_i) \vee \overline{\lambda}_i(\kappa_i \oplus k_i)$

$a_{i-1} = \lambda_i l_i \kappa_i k_i$

$m_i = \overline{\kappa}_i k_i \oplus \overline{\lambda}_i l_i$

$d_i = \alpha_i \oplus a_i \oplus m_i$

$b_{i-1} = a_i m_i \vee \alpha_i a_i \vee \alpha_i m_i$

$\sigma_i = b_i$ or $\sigma_i = d_i$

$s_i = d_i$ $\quad$ $s_i = b_i$

Table 2. Boolean Equations for Normalized Carry-Save Adders (Continued)

7. $\alpha_{i-1} = \lambda_i \kappa_i$

$a_{i-1} = \lambda_i \vee \kappa_i \vee l_i k_i$

$m_i = \bar{\lambda}_i l_i \oplus \bar{\kappa}_i k_i$

$d_i = m_i \oplus \bar{\alpha}_i a_i$

$b_{i-1} = \alpha_i \vee a_i m_i$

$\sigma_i = b_i d_i$

$s_i = b_i \vee d_i$

8. $\alpha_{i-1} = \lambda_i \bar{l}_i (\bar{\kappa}_i \vee \bar{k}_i) \vee \bar{\lambda}_i \kappa_i k_i \vee \lambda_i l_i \kappa_i k_i$

$a_{i-1} = \lambda_i l_i$ or $a_{i-1} = \kappa_i k_i$

$m_i = \lambda_i \oplus \kappa_i$

$d_i = \overline{\alpha_i \oplus m_i}$

$b_{i-1} = \alpha_i a_i \vee \bar{\alpha}_i m_i$

$\sigma_i = \overline{b_i \oplus d_i}$

$s_i = b_i$ or $s_i = d_i$

9. $\alpha_{i-1} = (\bar{\kappa}_i \bar{k}_i \oplus \bar{\lambda}_i \bar{l}_i) \vee \kappa_i (k_i \oplus \lambda_i) \vee \lambda_i (l_i \oplus \kappa_i)$

$a_{i-1} = \bar{\kappa}_i \bar{k}_i \lambda_i l_i \vee \bar{\kappa}_i k_i (\overline{\lambda_i \oplus l_i}) \vee \kappa_i \bar{\lambda}_i (k_i \oplus l_i)$

$m_i = (\bar{\kappa}_i k_i \oplus \bar{\lambda}_i l_i) \vee \lambda_i \bar{l}_i k_i$

$d_i = m_i \oplus \bar{\alpha}_i a_i$

$b_{i-1} = \bar{a}_i \bar{\alpha}_i \vee a_i m_i \vee \alpha_i a_i$

$\sigma_i = \overline{b_i \oplus d_i}$ or $\sigma_i = \bar{b}_i \bar{d}_i$

$s_i = b_i \vee d_i$ $\quad$ $s_i = b_i \oplus d_i$

Table 2. Boolean Equations for Normalized Carry-Save Adders (Continued)

4. SYMMETRIC ADDER AND SUBTRACTER

The symmetric adder and subtracter may be derived from the normalized adder by adding a constant to the input digit set and the output digit set, namely -1. Thus the values which may be assumed by the variables are as follows:

	$a_i^*, l_i^*, k_i^*, s_i^*$	b_i, m_i	d_i
SYMMETRIC ADDER	$\{\bar{1},0,1\}$	$\{0,1\}$	$\{\bar{1},0\}$
SYMMETRIC SUBTRACTER	$\{\bar{1},0,1\}$	$\{\bar{1},0\}$	$\{0,1\}$

The format assignments for the designs are given in Table 3. Tables 4 and 5 give the Boolean equations for the symmetric adder

State	Digital Values								
0 0	0	0	0	0	D.C.	0	0	1	$\bar{1}$
0 1	1	1	1	1	1	1	1	1	1
1 0	$\bar{1}$	D.C.	0	$\bar{1}$	0	1	$\bar{1}$	0	0
1 1	0	$\bar{1}$	$\bar{1}$	D.C.	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$	$\bar{1}$
Design Number	1	2	3	4	5	6	7	8	9

Table 3. Format Assignments for Symmetric Adder and Subtracter

procedures A and B. Tables 6 and 7 give the Boolean equations for the symmetric subtracter procedures A and B.

In Table 4 designs 2, 4, 5 and 7 appear to be minimal in the sense that they use the fewest numbers of gates. Designs 2, 4 and 5 have don't care entries in their format assignments.

Design 3, however, is probably of the most practical value since it allows digitwise complementation by complementing the first or sign bit of each digit. Thus, forming the negative value of a number is accomplished simply by complementing the sign bit of every digit.

In Table 5 designs 2, 4 and 5 appear to be minimal while in Tables 6 and 7 designs 2, 4, 5 and 7 appear to be minimal. Again the same comment about design 3 holds.

It is interesting to note that in general the results of procedure A are more complicated than those of procedure B. This is due to the constraint that the input and output formats of the box designed in procedure A be the same.

1. $\alpha_{i-1} = \kappa_i \bar{k}_i \bar{l}_i \vee \bar{\lambda}_i \bar{l}_i \bar{\kappa}_i \vee \bar{\kappa}_i \bar{k}_i \bar{\lambda}_i \vee \bar{\kappa}_i \bar{k}_i l_i$

   $a_{i-1} = \bar{\kappa}_i \bar{\lambda}_i \vee \bar{\lambda}_i \bar{l}_i \vee (k_i \oplus l_i) \vee \bar{l}_i \bar{\kappa}_i$

   $m_i = \kappa_i \oplus k_i \oplus \lambda_i \oplus l_i$

   $d_i = \alpha_i \oplus a_i \oplus m_i$

   $b_{i-1} = a_i m_i \vee \bar{\alpha}_i a_i \vee \bar{\alpha}_i m_i$

   $\sigma_i = d_i$ or $\sigma_i = \bar{b}_i$

   $s_i = b_i$ $\quad$ $s_i = \bar{d}_i$

2. $\alpha_{i-1} = \lambda_i \bar{k}_i \vee \kappa_i \bar{l}_i \vee \lambda_i \kappa_i$

   $a_{i-1} = \lambda_i \bar{k}_i \vee \kappa_i \bar{l}_i \vee \lambda_i \kappa_i \vee \bar{\lambda}_i l_i \bar{\kappa}_i k_i$

   $m_i = l_i \oplus k_i$

   $d_i = a_i \oplus m_i$

   $b_{i-1} = \bar{\alpha}_i (a_i \vee m_i)$

   $\sigma_i = \bar{b}_i d_i$

   $s_i = b_i \oplus d_i$

3. $\alpha_{i-1} = \lambda_i \vee \kappa_i$

   $a_{i-1} = \bar{\lambda}_i k_i (l_i \oplus \kappa_i) \vee \lambda_i \kappa_i k_i \vee \lambda_i l_i (\kappa_i \vee \bar{k}_i)$

   $m_i = l_i \oplus k_i$

   $d_i = a_i \oplus m_i$

   $b_{i-1} = \bar{\alpha}_i a_i \vee \bar{a}_i m_i$

   $\sigma_i = d_i$ or $\sigma_i = \bar{b}_i$

   $s_i = b_i \oplus d_i$

Table 4. Boolean Equations for Symmetric Carry-Save Adders

4. $\alpha_{i-1} = \lambda_i \overline{k}_i \vee \kappa_i \overline{l}_i \vee \lambda_i \kappa_i$

$a_{i-1} = l_i k_i$

$m_i = \overline{\lambda}_i \overline{l}_i \oplus \overline{\kappa}_i \overline{k}_i$

$d_i = m_i \oplus (a_i \vee \alpha_i)$

$b_{i-1} = a_i \vee \overline{\alpha}_i m_i$

$\sigma_i = \overline{b}_i d_i$

$s_i = b_i \overline{d}_i$

5. $\alpha_{i-1} = \lambda_i \vee \kappa_i$

$a_{i-1} = \lambda_i \kappa_i (k_i \vee l_i) \vee \overline{\lambda}_i \overline{\kappa}_i$

$m_i = l_i \oplus k_i$

$d_i = a_i \oplus m_i$

$b_{i-1} = \overline{\alpha}_i \vee \overline{a}_i m_i$

$\sigma_i = \overline{b}_i \vee d_i$

$s_i = b_i \oplus d_i$

6. $\alpha_{i-1} = \kappa_i k_i \vee \overline{\lambda}_i l_i (\kappa_i \vee k_i)$

$a_{i-1} = \lambda_i l_i \vee \lambda_i (\kappa_i + k_i)$

$m_i = \overline{\lambda}_i \overline{l}_i \oplus \overline{\kappa}_i \overline{k}_i$

$d_i = m_i \oplus (a_i \vee \alpha_i)$

$b_{i-1} = \overline{\alpha}_i a_i \vee \overline{a}_i m_i \vee \overline{a}_i \alpha_i$

$\sigma_i = \overline{b}_i d_i$ or $\sigma_i = b_i \oplus d_i$

$s_i = b_i \oplus d_i$ $\quad$ $s_i = \overline{b}_i d_i$

Table 4. Boolean Equations for Symmetric Carry-Save Adders (Continued)

7. $\alpha_{i-1} = \lambda_i \kappa_i \vee \lambda_i \overline{k}_i \vee \kappa_i \overline{l}_i$

$a_{i-1} = \overline{\lambda}_i l_i \overline{\kappa}_i k_i$

$m_i = \overline{\lambda}_i \overline{l}_i \oplus \overline{\kappa}_i \overline{k}_i$

$d_i = m_i \oplus (a_i \vee \alpha_i)$

$b_{i-1} = \overline{\alpha}_i (a_i \vee m_i)$

$\sigma_i = \overline{b}_i d_i$

$s_i = b_i \overline{d}_i$

8. $\alpha_{i-1} = \lambda_i \vee \kappa_i$

$a_{i-1} = \lambda_i \kappa_i (l_i \vee k_i)$

$m_i = \lambda_i \overline{l}_i \oplus \kappa_i \overline{k}_i$

$d_i = m_i \oplus (a_i \vee \overline{\alpha}_i)$

$b_{i-1} = \overline{\alpha}_i \vee \overline{a}_i m_i$

$\sigma_i = \overline{b}_i \vee d_i$

$s_i = \overline{b}_i d_i$

9. $\alpha_{i-1} = \kappa_i \overline{k}_i \vee (\overline{\lambda_i \oplus l_i}) (\kappa_i \vee \overline{k}_i) \vee \overline{\kappa}_i k_i \overline{\lambda}_i l_i$

$a_{i-1} = (\overline{\kappa_i \oplus k_i}) \vee (\overline{\lambda_i \oplus l_i}) \vee k_i l_i$

$m_i = \lambda_i \overline{l}_i \oplus \kappa_i \overline{k}_i$

$d_i = m_i \oplus (a_i \vee \overline{\alpha}_i)$

$b_{i-1} = \overline{\alpha}_i a_i \vee \alpha_i \overline{a}_i m_i$

$\sigma_i = \overline{b_i \oplus d_i}$ or $\sigma_i = \overline{b}_i \vee d_i$

$s_i = b_i \overline{d}_i$ $\qquad$ $s_i = b_i \oplus d_i$

Table 4. Boolean Equations for Symmetric Carry-Save Adders (Continued)

1. $m_i = \kappa_i \oplus \lambda_i \oplus l_i + k_i$

$d_{i-1} = m_{i-1} \oplus (\bar{\kappa}_i \bar{k}_i \vee \lambda_i \bar{l}_i \vee \bar{\kappa}_i k_i (\bar{\lambda}_i \vee l_i) \vee \bar{\lambda}_i l_i (\bar{\kappa}_i \vee k_i))$

$b_{i-2} = \kappa_i \bar{k}_i \lambda_i l_i \vee m_{i-1}(\lambda_i l_i \vee \kappa_i \bar{k}_i \vee \bar{l}_i \bar{k}_i \vee \kappa_i \bar{\lambda}_i)$

$\sigma_i = d_i$ or $\sigma_i = \bar{b}_i$

$s_i = b_i$ $\quad$ $s_i = \bar{d}_i$

2. $m_i = k_i \oplus l_i$

$d_{i-1} = \kappa_i \lambda_i \vee \bar{m}_{i-1} \bar{\kappa}_i \bar{\lambda}_i (k_i \vee l_i) \vee m_{i-1}(\lambda_i \vee \bar{l}_i \kappa_i \vee \bar{l}_i \bar{k}_i)$

$b_{i-2} = \kappa_i \lambda_i \vee m_{i-1}(\lambda_i \vee \kappa_i \vee \bar{l}_i \bar{k}_i)$

$\sigma_i = \bar{b}_i d_i$

$s_i = b_i \oplus d_i$

3. $m_i = k_i \oplus l_i$

$d_{i-1} = m_{i-1} \oplus (\bar{\kappa}_i \bar{\lambda}_i l_i \vee \bar{k}_i \bar{\lambda}_i l_i \vee \bar{\kappa}_i k_i \bar{l}_i \vee \kappa_i k_i \lambda_i l_i)$

$b_{i-2} = \kappa_i k_i \lambda_i l_i \vee m_{i-1}(\kappa_i k_i \vee \lambda_i l_i \vee \bar{k}_i \bar{l}_i)$

$\sigma_i = d_i$ or $\sigma_i = \bar{b}_i$

$s_i = b_i \oplus d_i$

4. $m_i = \bar{\kappa}_i \bar{k}_i \oplus \bar{\lambda}_i \bar{l}_i$

$d_{i-1} = \bar{m}_{i-1}(\bar{\kappa}_i l_i \vee k_i \bar{\lambda}_i \vee \kappa_i \lambda_i) \vee m_{i-1}(\kappa_i \oplus \lambda_i)$

$b_{i-2} = \kappa_i \lambda_i \vee m_{i-1}(\kappa_i \vee \lambda_i \vee \bar{k}_i \bar{l}_i)$

$\sigma_i = \bar{b}_i d_i$

$s_i = b_i \bar{d}_i$

Table 5. Boolean Equations for Symmetric Carry-Save Adders

5. $m_i = k_i \oplus l_i$

$$d_{i-1} = \overline{m}_{i-1}(\kappa_i k_i \lambda_i l_i \vee \overline{\kappa}_i \overline{\lambda}_i \vee \overline{\kappa}_i \overline{l}_i \vee \overline{k}_i \overline{\lambda}_i)$$
$$\vee\, m_{i-1}(\kappa_i \overline{l}_i \vee \overline{\kappa}_i \lambda_i l_i \vee \kappa_i k_i \overline{l}_i \vee \kappa_i k_i \overline{\lambda}_i)$$
$$b_{i-2} = \kappa_i k_i \lambda_i l_i \vee m_{i-1}(\kappa_i k_i \vee \lambda_i l_i \vee \kappa_i \lambda_i)$$

$$\sigma_i = \overline{b}_i \vee d_i$$
$$s_i = b_i \oplus d_i$$

6. $m_i = \overline{\kappa}_i \overline{k}_i \oplus \overline{\lambda}_i \overline{l}_i$

$$d_{i-1} = m_{i-1} \oplus ((\overline{\lambda}_i \vee \overline{l}_i)\ (\kappa_i \oplus k_i) \vee \overline{k}_i(\lambda_i \oplus l_i) \vee \kappa_i k_i \lambda_i l_i)$$
$$b_{i-2} = \kappa_i k_i \lambda_i l_i \vee m_{i-1}(\kappa_i k_i \vee \lambda_i l_i \vee \overline{\kappa}_i \overline{k}_i \overline{\lambda}_i \overline{l}_i)$$

$$\sigma_i = \overline{b}_i d_i \qquad \qquad \sigma_i = b_i \oplus d_i$$
or
$$s_i = b_i \oplus d_i \qquad \qquad s_i = \overline{b}_i d_i$$

7. $m_i = k_i \overline{\lambda}_i \overline{l}_i \vee \overline{\kappa}_i \overline{k}_i \overline{l}_i$

$$d_{i-1} = \overline{m}_{i-1}(\overline{\kappa}_i k_i \overline{\lambda}_i \vee \overline{\kappa}_i \overline{\lambda}_i l_i \vee \kappa_i \lambda_i)$$
$$\vee\, m_{i-1}((\kappa_i \oplus \lambda_i) \vee \overline{\kappa}_i \overline{k}_i \overline{l}_i)$$
$$b_{i-2} = \kappa_i \lambda_i \vee m_{i-1}(\kappa_i \vee \lambda_i \vee \overline{k}_i \overline{l}_i)$$

$$\sigma_i = \overline{b}_i d_i$$
$$s_i = b_i \overline{d}_i$$

Table 5. Boolean Equations for Symmetric Carry-Save Adders (Continued)

8. $$m_i = \kappa_i\bar{k}_i \oplus \lambda_i\bar{l}_i$$

$$d_{i-1} = m_{i-1} \oplus (\bar{\kappa}_i\bar{\lambda}_i \vee \bar{k}_i\bar{\lambda}_i \vee \bar{\kappa}_i\bar{l}_i \vee \kappa_i k_i \lambda_i l_i)$$

$$b_{i-2} = \kappa_i k_i \lambda_i l_i \vee m_{i-1}(\kappa_i k_i \vee \kappa_i \lambda_i \vee \bar{k}_i \lambda_i l_i)$$

$$\sigma_i = \bar{b}_i \vee d_i$$

$$s_i = \bar{b}_i d_i$$

9. $$m_i = \kappa_i\bar{k}_i \oplus \lambda_i\bar{l}_i$$

$$d_{i-1} = m_{i-1} \oplus (\overline{\kappa_i \oplus k_i \oplus \lambda_i \oplus l_i})$$

$$b_{i-2} = \lambda_i l_i(\kappa_i \vee \bar{k}_i) \vee \bar{\lambda}_i\bar{l}_i(\overline{\kappa_i \oplus k_i}) \vee m_i((\overline{\kappa_i \oplus k_i}) \vee (\overline{\lambda_i \oplus l_i}))$$

$$\sigma_i = \overline{b_i \oplus d_i} \qquad \text{or} \qquad \sigma_i = \bar{b}_i \vee d_i$$

$$s_i = b_i\bar{d}_i \qquad \qquad s_i = b_i \oplus d_i$$

Table 5. Boolean Equations for Symmetric Carry-Save Adders (Continued)

1. $\alpha_{i-1} = \overline{\lambda}_i l_i \vee \overline{\kappa}_i k_i \vee l_i \kappa_i \overline{k}_i$

$a_{i-1} = \lambda_i \overline{l}_i \vee \kappa_i \overline{k}_i \vee (\overline{\lambda}_i l_i \oplus \overline{\kappa}_i k_i)$

$m_i = \kappa_i \oplus k_i \oplus \lambda_i \oplus l_i$

$d_i = \alpha_i \oplus a_i \oplus m_i$

$b_{i-1} = \overline{a}_i m_i \vee \overline{a}_i \alpha_i \vee \alpha_i m_i$

$\sigma_i = b_i$ or $\sigma_i = \overline{d}_i$

$s_i = d_i$ $\quad$ $s_i = \overline{b}_i$

2. $\alpha_{i-1} = \lambda_i \kappa_i$

$a_{i-1} = \lambda_i \kappa_i \vee \overline{\lambda}_i \overline{\kappa}_i (l_i \vee k_i)$

$m_i = l_i \oplus k_i$

$d_i = a_i \oplus m_i$

$b_{i-1} = \alpha_i \vee \overline{a}_i m_i$

$\sigma_i = b_i \overline{d}_i$

$s_i = b_i \oplus d_i$

3. $\alpha_{i-1} = \lambda_i \kappa_i$ or $\lambda_i l_i$ or $\kappa_i k_i$

$a_{i-1} = \overline{\lambda}_i l_i (\overline{\kappa}_i \vee \overline{k}_i) \vee \overline{\kappa}_i k_i \overline{l}_i \vee \lambda_i l_i \kappa_i k_i$

$m_i = l_i \oplus k_i$

$d_i = a_i \oplus m_i$

$b_{i-1} = \alpha_i a_i \vee \overline{a}_i m_i$

$\sigma_i = b_i$ or $\sigma_i = \overline{d}_i$

$s_i = b_i \oplus d_i$

Table 6. Boolean Equations for Symmetric Borrow-Save Subtracters

4. $\alpha_{i-1} = \lambda_i \kappa_i$

$a_{i-1} = l_i \overline{\kappa}_i \vee \overline{\lambda}_i k_i$

$m_i = \overline{\lambda}_i \overline{l}_i \oplus \overline{\kappa}_i \overline{k}_i$

$d_i = m_i \oplus (\alpha_i \vee a_i)$

$b_{i-1} = \alpha_i \vee \overline{a}_i m_i$

$\sigma_i = b_i \overline{d}_i$

$s_i = \overline{b}_i d_i$

5. $\alpha_{i-1} = \lambda_i l_i \vee \kappa_i k_i \vee \lambda_i \kappa_i$

$a_{i-1} = \kappa_i (\overline{\lambda}_i \vee \overline{l}_i) \vee \overline{\kappa}_i \overline{k}_i \vee \lambda_i l_i \kappa_i k_i$

$m_i = l_i \oplus k_i$

$d_i = a_i \oplus m_i$

$b_{i-1} = \alpha_i (a_i \vee m_i)$

$\sigma_i = b_i \vee \overline{d}_i$

$s_i = b_i \oplus d_i$

6. $\alpha_{i-1} = \kappa_i k_i (\overline{\lambda_i \oplus l_i}) \vee \lambda_i l_i \overline{\kappa}_i \overline{k}_i$

$a_{i-1} = \kappa_i k_i (\overline{\lambda_i \oplus l_i}) \vee \lambda_i l_i \overline{\kappa}_i \overline{k}_i \vee (\kappa_i \oplus k_i)\ (\lambda_i \oplus l_i)$

$m_i = \overline{\lambda}_i \overline{l}_i \oplus \overline{\kappa}_i \overline{k}_i$

$d_i = m_i \oplus (\alpha_i \vee a_i)$

$b_{i-1} = \alpha_i a_i \vee \overline{\alpha}_i \overline{a}_i m_i$

$\sigma_i = b_i \overline{d}_i$ or $\sigma_i = b_i \oplus d_i$

$s_i = b_i \oplus d_i$ $\quad$ $s_i = b_i \overline{d}_i$

Table 6. Boolean Equations for Symmetric Borrow-Save Subtracters (Continued)

7. $\alpha_{i-1} = \lambda_i \kappa_i$

$a_{i-1} = \bar{\lambda}_i \bar{\kappa}_i (l_i \vee k_i)$

$m_i = \bar{\lambda}_i \bar{l}_i \oplus \bar{\kappa}_i \bar{k}_i$

$d_i = m_i \oplus (\alpha_i \vee a_i)$

$b_{i-1} = \alpha_i \vee \bar{a}_i m_i$

$\sigma_i = b_i \bar{d}_i$

$s_i = \bar{b}_i d_i$

8. $\alpha_{i-1} = \lambda_i l_i \vee \kappa_i k_i \vee \lambda_i \kappa_i$

$a_{i-1} = \lambda_i l_i \kappa_i k_i$

$m_i = \lambda_i \bar{l}_i \oplus \kappa_i \bar{k}_i$

$d_i = m_i \oplus (\bar{\alpha}_i \vee a_i)$

$b_{i-1} = \alpha_i (a_i \vee m_i)$

$\sigma_i = b_i \vee \bar{d}_i$

$s_i = b_i \bar{d}_i$

9. $\alpha_{i-1} = \bar{k}_i \vee \kappa_i \vee \lambda_i \vee \bar{l}_i$

$a_{i-1} = (\kappa_i \vee \bar{k}_i)(\overline{\lambda_i \oplus l_i}) \vee \bar{l}_i (\overline{\kappa_i \oplus k_i}) \vee \bar{\kappa}_i k_i \bar{\lambda}_i l_i$

$m_i = \lambda_i \bar{l}_i \oplus \kappa_i \bar{k}_i$

$d_i = m_i \oplus (\bar{\alpha}_i \vee a_i)$

$b_{i-1} = \alpha_i a_i \vee a_i m_i \vee \bar{\alpha}_i \bar{a}_i$

$\sigma_i = b_i \vee \bar{d}_i$ or $\sigma_i = \overline{b_i \oplus d_i}$

$s_i = b_i \oplus d_i$ $\quad$ $s_i = \bar{b}_i d_i$

Table 6. Boolean Equations for Symmetric Borrow-Save Subtracters (Continued)

1. $m_i = \kappa_i \oplus k_i \oplus \lambda_i \oplus l_i$

$d_i = m_{i-1} \oplus (\kappa_i \bar{k}_i \bar{l}_i \vee \kappa_i \bar{k}_i \lambda_i \vee \bar{k}_i \lambda_i \bar{l}_i \vee \kappa_i \lambda_i \bar{l}_i \vee \bar{\kappa}_i k_i \bar{\lambda}_i l_i)$

$b_{i-2} = \bar{\kappa}_i k_i \bar{\lambda}_i l_i \vee m_{i-1}(\bar{\lambda}_i l_i \vee \bar{\kappa}_i k_i \vee k_i \bar{l}_i \vee \bar{\kappa}_i \bar{\lambda}_i \vee \bar{\kappa}_i l_i)$

$\sigma_i = b_i$ or $\sigma_i = \bar{d}_i$

$s_i = d_i$ $\quad$ $s_i = \bar{b}_i$

2. $m_i = k_i \oplus l_i$

$d_{i-1} = m_{i-1} \oplus (\kappa_i \lambda_i \vee \bar{k}_i \bar{l}_i \vee \bar{k}_i \lambda_i \vee \bar{\kappa}_i k_i \bar{\lambda}_i l_i)$

$b_{i-2} = \bar{\kappa}_i k_i \bar{\lambda}_i l_i \vee m_{i-1}(\bar{\lambda}_i l_i \vee \bar{\kappa}_i k_i \vee \bar{\kappa}_i \bar{\lambda}_i)$

$\sigma_i = b_i \bar{d}_i$

$s_i = b_i \oplus d_i$

3. $m_i = k_i \oplus l_i$

$d_{i-1} = m_{i-1} \oplus (\kappa_i \lambda_i l_i \vee \bar{k}_i \lambda_i l_i \vee \kappa_i k_i \lambda_i \vee \bar{\kappa}_i k_i \bar{\lambda}_i l_i)$

$b_{i-2} = \bar{\kappa}_i k_i \bar{\lambda}_i l_i \vee \bar{m}_{i-1} \kappa_i k_i \lambda_i l_i \vee m_{i-1}(\bar{\kappa}_i k_i \vee \bar{\lambda}_i l_i \vee k_i l_i)$

$\sigma_i = b_i$ or $\sigma_i = \bar{d}_i$

$s_i = b_i \oplus d_i$

4. $m_i = \bar{\kappa}_i \bar{k}_i \oplus \bar{\lambda}_i \bar{l}_i$

$d_{i-1} = m_{i-1} \oplus (k_i l_i \vee \kappa_i \bar{l}_i \vee \bar{k}_i \lambda_i)$

$b_{i-2} = k_i l_i \vee m_{i-1}(k_i \vee l_i \vee \bar{\kappa}_i \bar{\lambda}_i)$

$\sigma_i = b_i \bar{d}_i$

$s_i = \bar{b}_i d_i$

Table 6. Boolean Equations for Symmetric Borrow-Save Subtracters (Continued)

5. $m_i = k_i \oplus l_i$

$d_{i-1} = \bar{m}_{i-1} \oplus ((\kappa_i \oplus \lambda_i) \vee \bar{k}_i \bar{l}_i)$

$b_{i-2} = \bar{\kappa}_i \bar{\lambda}_i \vee m_{i-1}(\bar{\kappa}_i \vee \bar{\lambda}_i \vee \bar{k}_i \bar{l}_i)$

$\sigma_i = b_i \vee \bar{d}_i$

$s_i = b_i \oplus d_i$

6. $m_i = \bar{\kappa}_i \bar{k}_i \oplus \bar{\lambda}_i \bar{l}_i$

$d_{i-1} = m_{i-1} \oplus ((\kappa_i \oplus k_i)(\lambda_i \oplus l_i) \vee \kappa_i k_i (\overline{\lambda_i \oplus l_i}) \vee \bar{\kappa}_i k_i \bar{\lambda}_i l_i)$

$b_{i-2} = (\kappa_i \oplus k_i)(\lambda_i \oplus l_i) \vee m_{i-1}((\lambda_i \oplus l_i)$
$\quad \vee (\kappa_i \oplus k_i) \vee \bar{\kappa}_i \bar{\lambda}_i)$

$\sigma_i = b_i \bar{d}_i$ or $\sigma_i = b_i \oplus d_i$

$s_i = b_i \oplus d_i$ $\quad$ $s_i = b_i \bar{d}_i$

7. $m_i = \bar{\kappa}_i \bar{k}_i \oplus \bar{\lambda}_i \bar{l}_i$

$d_{i-1} = m_{i-1} \oplus (\kappa_i \lambda_i \vee \bar{k}_i \lambda_i \vee \kappa_i \bar{l}_i \vee \bar{\kappa}_i k_i \bar{\lambda}_i l_i)$

$b_{i-2} = \bar{\kappa}_i k_i \bar{\lambda}_i l_i \vee m_{i-1}(\bar{\kappa}_i k_i \vee \bar{\lambda}_i l_i \vee \bar{\kappa}_i \bar{\lambda}_i)$

$\sigma_i = b_i \bar{d}_i$

$s_i = \bar{b}_i d_i$

8. $m_i = \kappa_i \bar{k}_i \oplus \lambda_i \bar{l}_i$

$d_{i-1} = \bar{m}_{i-1} \bar{\kappa}_i \bar{\lambda}_i \vee m_{i-1}((\kappa_i \oplus \lambda_i) \vee \kappa_i \bar{k}_i \lambda_i \bar{l}_i)$

$b_{i-2} = \bar{\kappa}_i \bar{\lambda}_i \vee m_{i-1}(\bar{\kappa}_i \vee \bar{\lambda}_i \vee \kappa_i \bar{k}_i \lambda_i \bar{l}_i)$

$\sigma_i = b_i \vee \bar{d}_i$

$s_i = b_i \bar{d}_i$

Table 6. Boolean Equations for Symmetric Borrow-Save Subtracters
(Continued)

9. $m_i = \kappa_i \bar{k}_i \oplus \lambda_i \bar{l}_i$

$d_{i-1} = m_{i-1} \oplus (\bar{\kappa}_i k_i \bar{\lambda}_i l_i \vee (\lambda_i \vee \bar{l}_i)(\overline{\kappa_i \oplus k_i}) \vee \kappa_i(\overline{\lambda_i \oplus l_i}))$

$b_{i-2} = \bar{\kappa}_i k_i \bar{\lambda}_i l_i \vee m_{i-1}(\bar{\kappa}_i k_i \vee \bar{\lambda}_i l_i \vee \kappa_i \bar{k}_i \lambda_i \bar{l}_i)$

$$\begin{array}{lcl} \sigma_i = b_i \vee \bar{d}_i & & \sigma_i = \overline{b_i \oplus d_i} \\ & \text{or} & \\ s_i = b_i \oplus d_i & & s_i = \bar{b}_i d_i \end{array}$$

Table 6. Boolean Equations for Symmetric Borrow-Save Subtracters (Continued)

5. COUPLED DON'T CARES

One of the interesting problems encountered in the design of redundant adders is the coupled don't care. This condition arises when a digital value has two representations and the two variables used to represent the value are mutually complementary in the two representations. That is, if $\lambda_i l_i$ is one representation, the other is $\overline{\lambda}_i \overline{l}_i$. An example showing the coupled don't care condition is shown in Table 8, the truth table for the symmetric subtracter for design 1A.

λ	l	κ	k	α	a	b	Line
0	0	0	0	0/1	0/1	0	1
0	0	0	0	0	1	1	2
0	0	1	0	0/1	0/1	1	3
0	0	1	1	0/1	0/1	0	4
0	1	0	0	0	1	1	5
0	1	0	1	0	1	0	6
0	1	1	0	0/1	0/1	0	7
0	1	1	1	0	1	1	8
1	0	0	0	0/1	0/1	1	9
1	0	0	1	0/1	0/1	0	10
1	0	1	0	1	0	0	11
1	0	1	1	0/1	0/1	1	12
1	1	0	0	0/1	0/1	0	13
1	1	0	1	0	1	1	14
1	1	1	0	0/1	0/1	1	15
1	1	1	1	0/1	0/1	0	16

Table 8

Notice that the value for α_i in lines 1, 3, 4, 7, 9, 10, 12, 13, 15, and 16 may be chosen as either 0 or 1 but that the choice for

a_i is then constrained to be the same. It is readily apparent that the choice of don't cares to simplify α_i may make a_i more complicated. This problem is similar to the case of trying to minimize two functions simultaneously.

While no procedure has been discovered which will guarantee the absolute minimal functions, a procedure which gives all possible functions has been investigated.

The procedure begins by introducing a function ϕ consisting of all the don't cares chosen to be one. Then one minimizes the functions $f \vee \phi$ and $g \vee \phi$. The steps in the procedure are as follows:

i) Find all implicants of f and g.

ii) Find all prime implicants of $f \vee \phi_i$ which simplify the the expression for f.

iii) Using the ϕ_i determined in step ii) find all prime implicants of $g \vee \phi_i$.

iv) Find all minimal covers for f using the prime implicants of $f \vee \phi_i$.

v) Select cover from step iv) which gives minimal expression for g.

The functions from Table 8 will be used to illustrate the procedure. For convenience a Karnaugh map showing both functions is given on the following page.

	$\overline{Y}\,\overline{Z}$	$\overline{Y}\,Z$	$Y\,Z$	$Y\,\overline{Z}$
$\overline{W}\,\overline{X}$	0	1 f	3	2
$\overline{W}\,X$	4 f	5 f	7 f	6
$W\,X$	12	13 f	15	14
$W\,\overline{X}$	8	9	11	10 g

1. Implicants of f and g

f: 1,4,5,7,13

g: 10

2. Prime implicants of $f \vee \phi$ which give reduction of f

ϕ	$f \vee \phi$
0,2,3,6	0,1,2,3,4,5,6,7
6,12,14,15	4,5,6,7,12,13,14,15
0,8,9,12	0,1,4,5,8,9,12,13
3,9,11,15	1,3,5,7,9,11,13,15
0,2,3	0,1,2,3
0,8,12	0,4,8,12
0	0,1,4,5
0,2,6	0,2,4,6
9	1,5,9,13
3	1,3,5,7
3,9,11	1,3,9,11
2,3,6	2,3,6,7
3,11,15	3,7,11,15
6	4,5,6,7
12	4,5,12,13
6,12,14	4,6,12,14
15	5,7,13,15
6,14,15	6,7,14,15
12,14,15	12,13,14,15
9,11,15	9,11,13,15
8,9,12	8,9,12,13
0,8,9	0,1,8,9
Null	1,5
	4,5
	5,7
	5,13

3. Using ϕ determined in step 2 prime implicants of $g \vee \phi$

0,2,3,6	0,2; 2,3; 2,6; 2,10
6,12,14,15	6,14; 14,15; 12,14; 10,14
0,8,9,12	0,8; 8,12; 8,9; 8,10
3,9,11,15	3,11; 11,15; 9,11; 10,11
0,2,3	0,2; 2,3; 2,10
0,8,12	0,8; 0,12; 8,12; 8,10
0	0; 10
0,2,6	0,2; 2,6; 6,10
9	9; 10
3	3; 10
3,9,11	3,11; 9,11; 10,11
2,3,6	2,3; 2,6; 2,10
3,11,15	3,11; 3,15; 10,11
6	6; 10
12	12; 10
6,12,14	6,14; 12,14; 10,14
15	10; 15
6,14,15	6,14; 14,15; 10,14
12,14,15	12,14; 14,15; 10,14
9,11,15	9,11; 11,15; 10,11
8,9,12	8,9; 8,12; 8,10
0,8,9	0,8; 8,9; 8,10
Null	10

4. Covers for f using prime implicants of $f \vee \phi$

	Prime implicants of $f \vee \phi$	Implicants of f 1	4	5	7	13
R_1	0,1,2,3,4,5,6,7	X	X	X	X	
R_2	4,5,6,7,12,13,14,15		X	X	X	X
R_3	0,1,4,5,8,9,12,13	X	X	X		X
R_4	1,3,5,7,9,11,13,15	X			X	X
R_5	0,1,2,3	X				
R_6	0,4,8,12		X			
R_7	0,1,4,5	X	X	X		
R_8	0,2,4,6		X			
R_9	1,5,9,13	X		X		X
R_{10}	1,3,5,7	X		X	X	
R_{11}	1,3,9,11	X				
R_{12}	2,3,6,7				X	
R_{13}	3,7,11,15				X	
R_{14}	4,5,6,7		X	X	X	
R_{15}	4,5,12,13		X	X		X
R_{16}	4,6,12,14		X			
R_{17}	5,7,13,15			X	X	X
R_{18}	6,7,14,15				X	
R_{19}	12,13,14,15					X
R_{20}	9,11,13,15					X
R_{21}	8,9,12,13					X
R_{22}	0,1,8,9	X				
R_{23}	1,5	X		X		
R_{24}	4,5		X	X		
R_{25}	5,7			X	X	
R_{26}	5,13			X		X

$$f = (R_1 \vee R_3 \vee R_4 \vee R_5 \vee R_7 \vee R_9 \vee R_{10} \vee R_{11} \vee R_{22} \vee R_{23}) (R_1 \vee R_2 \vee R_3 \vee R_6 \vee R_7 \vee R_8 \vee R_{14} \vee R_{15} \vee R_{16} \vee R_{24}) (R_1 \vee R_2 \vee R_3 \vee R_7 \vee R_9 \vee R_{10} \vee R_{14} \vee R_{15} \vee R_{17} \vee R_{23} \vee R_{24} \vee R_{25} \vee R_{26}) (R_1 \vee R_2 \vee R_4 \vee R_{10} \vee R_{12} \vee R_{13} \vee R_{14} \vee R_{17} \vee R_{18} \vee R_{25}) (R_2 \vee R_3 \vee R_4 \vee R_9 \vee R_{15} \vee R_{17} \vee R_{19} \vee R_{20} \vee R_{21} \vee R_{26})$$

Obviously all possible covers which give a reduction of f are given by the above. The task remains to select the minimal ones. Many of the covers are not minimal and may be disregarded. Implicants R_1, R_2, R_3 and R_4 correspond to $\overline{W}$, X, $\overline{Y}$ and Z respectively. Minimal covers of f are formed by choosing any two of the above implicants. Thus there are 6 covers corresponding to R_1R_2, R_1R_3, R_1R_4, R_2R_3, R_2R_4 and R_3R_4. The functions for these covers are tabulated below.

Cover	f	g
R_1R_2	$\overline{W} \vee X$	$Y\,\overline{Z} \vee (Y \vee \overline{Z})\,(\overline{W \oplus X})$
R_1R_3	$\overline{W} \vee \overline{Y}$	$\overline{X}\,\overline{Z} \vee (\overline{X} \vee \overline{Z})\,(W \oplus Y)$
R_1R_4	$\overline{W} \vee Z$	$Y\,\overline{X} \vee (Y \vee \overline{X})\,(\overline{W \oplus Z})$
R_2R_3	$X \vee \overline{Y}$	$W\,\overline{Z} \vee (W \vee \overline{Z})\,(\overline{X \oplus Y})$
R_2R_4	$X \vee Z$	$W\,Y \vee (W \vee Y)\,(X \oplus Z)$
R_3R_4	$\overline{Y} \vee Z$	$W\,\overline{X} \vee (W \vee \overline{X})\,(\overline{Y \oplus Z})$

These functions are also the minimal functions as can be seen by checking the covers provided by choosing a more complicated function for f.

It is apparent that one may also apply this procedure to g first and find covers for f. This is not necessary, however, as shown by the following theorem (7).

Theorem: Given a solution of the form

$$f_i = f^1 \vee \phi_i^1 \text{ and } g_i = g^1 \vee \phi_i^1$$

there exists a solution of the form

$$f_j = \overline{g_i} \text{ and } g_j = \overline{f_i},$$

where f^1 and g^1 indicate the minterms where f and g are 1 respectively and ϕ_i^1 indicates the don't cares chosen to be 1.

Proof:

$$f_j = \overline{g_i} = \overline{g^1 \vee \phi_i^1} = g^0 \vee \phi_i^0 = f^1 \vee \phi_j^1$$

$$g_j = \overline{f_i} = \overline{f^1 \vee \phi_i^1} = f^0 \vee \phi_i^0 = g^1 \vee \phi_j^1,$$

where $\phi_j^1 = \phi_i^0$.

Thus, one need only find minimal functions starting with f, complement the results and obtain the functions which would be obtained starting with g. In the above example, therefore, there are actually twelve minimal solutions.

The case described in this example is very symmetric as are all the other cases encountered in the adder design. This is due to the symmetry of the problem and the format assignment.

The problem of complementary coupled don't cares (that is, α_i may be chosen as either 0 or 1 but then a_1 is constrained to be the complement of α_1) is solved in a similar manner. That is, one finds the covers for f and then picks the corresponding minimal function for the complement of g.

Note: The reader unfamiliar with the notation used in representing implicants and covers is referred to (4).

6. SUMMARY

This thesis has shown the steps used in the logical design of a class of limited carry-borrow propagation adders. Fifty four designs were presented, most of them for the first time. Some designs were pointed out as appearing to be minimal.

No logic drawings or gate counts were presented as this is a function entirely of implementation. These designs, in fact, lend themselves readily to Large Scale Integration since they are modular, fairly simple, and have only a few lines crossing module boundaries.

The primary use of these designs would probably be in a machine dedicated to total redundancy. In the case that this were not practical, the designs may be used for assimilation also. For example, in the case of the symmetric adder and symmetric subtracter the redundant number to be assimilated is applied to a_i and α_i, b_{i-1} is used as a propagating signal and is applied to input m_{i-1}, and the conventional (i.e. assimilated) result is available at d_i. This configuration is shown in Figure 6.

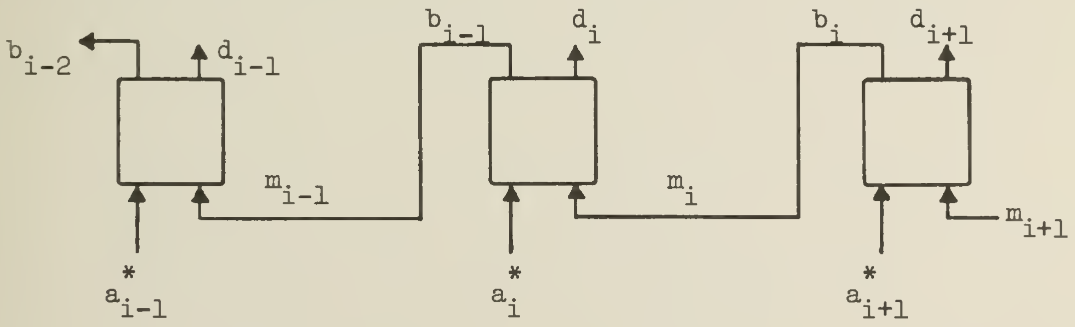

Figure 6. Adder Structure Used as Assimilator

Using Rohatsch's transformation theory, it is readily apparent that more levels may be added to the adder structure allowing more redundant inputs. For example, a fourth level may be added to the three level structure which will allow six redundant inputs. It is also evident that one can add a conventional input to the three level structure with no increase in cost. This was not done, however, since we were interested only in total redundancy.

Furthermore, the requirement that the format of all redundant interstage signals be the same is not necessary. In fact the recognition of this fact prompted design procedure B. As a result there are actually 729 possible designs which are simply all possible combinations of the three levels.

The most interesting area for future investigation is the statistical properties of the adders. The analysis of the structures has just begun (7,8) and has pointed out the difficulty of the process.

LIST OF REFERENCES

1. Atkins, D. E., "Arithmetic Unit of Illiac III: Simulation and Logical Design - Part I", Department of Computer Science, University of Illinois, Urbana, Illinois 61801, File No. 713, September, 1966.

2. Atkins, D. E., "The Theory and Implementation of SRT Division", Department of Computer Science, University of Illinois, Urbana, Illinois 61801, Report No. 230, June, 1967 (M.S. Thesis).

3. Avizienis, A., "Signed-Digit Number Representation for Fast Parallel Arithmetic", IRE Transactions on Electronic Computers, Vol. EC-10, No. 3, pp. 389-400, September, 1961.

4. Hohn, F. E., Applied Boolean Algebra, MacMillan Company, New York, 1966.

5. Robertson, J. E., "Increasing the Efficiency of Digital Computer Arithmetic Through Use of Redundancy", Notes for EE497B, University of Illinois, Urbana, Illinois 61801, Fall, 1964.

6. Robertson, J. E., "A Deterministic Procedure for the Design of Carry-Save Adders and Borrow - Save Subtracters", Department of Computer Science, University of Illinois, Urbana, Illinois 61801, Report No. 235, July, 1967.

7. Robertson, J. E., Private Communication, March, 1968.

8. Rohatsch, F. A., "A Study of Transformations Applicable to the Development of Limited Carry - Borrow Propagation Adders", Department of Computer Science, University of Illinois, Urbana, Illinois 61801, Report No. 226, June, 1967 (Ph.D. Thesis).